Wine quality prediction with Machine Learning

edX Capstone Final Project

Ciro B Rosa

21-Jan-2022

Introduction

This is the second project submitted by the author for the HarvardX Data Science course, under the Capstone module. It aims to predict quality of wine based on its chemical composition. The source dataset can be found on Kaggle, at the following link:

https://www.kaggle.com/yasserh/wine-quality-dataset

For ease of execution, all needed files can be downloaded from Github's author's page at:

https://github.com/cirobr/ds9-capstone-winequality.git

Project Contents

Dataset The dataset "WineQT.csv" is stored on folder "/dat. It has 1142 observations and 13 (thirteen) columns: one outcome (quality), eleven predictors, and one column (Id), which is merely a sequential numbering for the observations. All columns are named at first row.

Code Three files are presented in this project:

- File "project-code.R" contains the entire script for the project;
- File "project-nb.Rmd" is its correspondent notebook file;
- File "project-nb.pdf" is the notebook executed and stored in PDF format.

Project Description

Code starts with environment setup, then the dataset is read and cleaned up. Dataset is randomly split in three parts: Training, Testing, and Validation. The Validation split will not be touched up to the point the final model is chosen.

Next, a sequence of ML models are applied to classify the outcome "quality". Its efficiency is measured by the Accuracy calculated parameter and its correspondent Confusion Matrix. A table with all accuracies is generated at the end of each model, in order to compare efficiencies.

At the end, the Validation split will be run, in order to confirm the forecasted accuracy and if there has been overtraning of the model.

The project was developed and tested on a general purpose Linux Ubuntu 20.04 notebook, with both RStudio Version 1.4.1717 and R version 4.1.2 installed. It was also tested on a VM Linux Ubuntu 20.04 guest over a Windows 10 host. As such, code is expected to run smoothly on any machine with 8GB RAM.

Project Execution

```r
# suppress warnings
oldw <- getOption("warn")
options(warn = -1)

# install packages (if not already installed)
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
```

Environment setup

```
## Loading required package: ggplot2
```

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
## v purrr   0.3.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: matrixStats
##
## Attaching package: 'matrixStats'
## The following object is masked from 'package:dplyr':
##
##     count
```

```r
if(!require(nnet)) install.packages("nnet", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: nnet
```

```r
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: randomForest
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
if(!require(kknn)) install.packages("kknn", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: kknn

##
## Attaching package: 'kknn'

## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
# libraries
library(ggplot2)
library(tidyverse)
library(caret)
library(matrixStats)
library(nnet)          # multinom model
library(randomForest)  # rf model
library(kknn)          # kknn model

# global variables
numberOfDigits <- 5
options(digits = numberOfDigits)
proportionTestSet <- 0.20
control <- trainControl(method = "cv",   # 10-fold cross validation configuration
                        number = 10,
                        p = .9,
                        allowParallel = TRUE)

# global functions
plot_optimization_chart <- function(model, chart_title){
  m <- model %>%
    ggplot(highlight = TRUE) +
    ggtitle(chart_title)
  print(m)
  model$bestTune
  model$finalModel
}
```

Next, the main dataset is read:

```
# read dataset from csv
# source of dataset: https://www.kaggle.com/yasserh/wine-quality-dataset
wineDF <- read_csv(file = "./dat/WineQT.csv") %>% as_tibble()
```

```
## Rows: 1143 Columns: 13

## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## dbl (13): fixed acidity, volatile acidity, citric acid, residual sugar, chlo...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
glimpse(wineDF)
```

```
## Rows: 1,143
## Columns: 13
## $ `fixed acidity`        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, 7.8, 6~
## $ `volatile acidity`     <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660, 0.600~
## $ `citric acid`          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06, 0.00,~
## $ `residual sugar`       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2.0, 1.~
## $ chlorides              <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075, 0.069~
## $ `free sulfur dioxide`  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 15, 16, 9, 3~
## $ `total sulfur dioxide` <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 65, 59, 29,~
## $ density                <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0.9978,~
## $ pH                     <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30, 3.39,~
## $ sulphates              <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46, 0.47,~
## $ alcohol                <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, 9.5, 9~
## $ quality                <dbl> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 7, 6, 5, 5,~
## $ Id                     <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 16, 19, ~
```

A basic cleanup is also done, as the original dataset is already provided in very good condition:

```
# remove column "Id" (not used) and make outcome "quality" as first column
wineDF <- wineDF %>% select(-Id) %>% relocate(quality)

# on column names, replace spaces by underline "_"
names(wineDF) <- gsub(" ", "_", names(wineDF))
names(wineDF)
```

```
##  [1] "quality"              "fixed_acidity"        "volatile_acidity"
##  [4] "citric_acid"          "residual_sugar"       "chlorides"
##  [7] "free_sulfur_dioxide"  "total_sulfur_dioxide" "density"
## [10] "pH"                   "sulphates"            "alcohol"
```

```
# check for NA's
sum(is.na(wineDF))
```

```
## [1] 0
```

```
# display the cleaned dataset
head(wineDF)
```

```
## # A tibble: 6 x 12
##   quality fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
##     <dbl>         <dbl>            <dbl>       <dbl>          <dbl>     <dbl>
## 1       5           7.4             0.7         0               1.9     0.076
## 2       5           7.8             0.88        0               2.6     0.098
## 3       5           7.8             0.76        0.04            2.3     0.092
## 4       6          11.2             0.28        0.56            1.9     0.075
## 5       5           7.4             0.7         0               1.9     0.076
## 6       5           7.4             0.66        0               1.8     0.075
## # ... with 6 more variables: free_sulfur_dioxide <dbl>,
## #   total_sulfur_dioxide <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
```

```
## #   alcohol <dbl>
```

Now, it is time to split the main dataset as follows: validationSet, testSet, and trainSet:

```
# split validation set for the final step (20%)
set.seed(1, sample.kind="Rounding")
testIndex <- createDataPartition(y = wineDF$quality,
                                 times = 1,
                                 p = proportionTestSet,
                                 list = FALSE)
validationSet <- wineDF[testIndex,]

# from the remaining dataset, split train and test sets (80 / 20%)
mainSet <- wineDF[-testIndex,]

set.seed(2, sample.kind="Rounding")
testIndex <- createDataPartition(y = mainSet$quality,
                                 times = 1,
                                 p = proportionTestSet,
                                 list = FALSE)
testSet  <- mainSet[testIndex,]
trainSet <- mainSet[-testIndex,]
```

Data exploration First of all, let's quickly check which wine quality rankings are present on the dataset:

```
# check for the different wine quality classes on the trainSet
qualityClasses <- unique(trainSet$quality) %>% sort()
qualityClasses
```

```
## [1] 3 4 5 6 7 8
```

Secondly, let's check a histogram for the distribution of wine quality. At the same time, we will check if, for each quality category, the data split respected the proportion for each of the classes at both trainSet and testSet:

```
# check for stratification of train / test split
p1 <- trainSet %>%
  group_by(quality) %>%
  summarize(qty = n()) %>%
  mutate(split = 'trainSet')

p2 <- testSet %>%
  group_by(quality) %>%
  summarize(qty = n()) %>%
  mutate(split = 'testSet')

p <- bind_rows(p1, p2) %>% group_by(split)
p %>% ggplot(aes(quality, qty, fill = split)) +
  geom_bar(stat="identity", position = "dodge") +
  ggtitle("Stratification of Testset / Trainset split") +
  scale_x_continuous(breaks = qualityClasses)
```

On the above chart, it is seen that the amount of training data for categories 4-7 is much higher than for categories 3 and 8. This fact could lead to difficulties on the model forecast development, as the categories with little data may not be predicted with good accuracy.

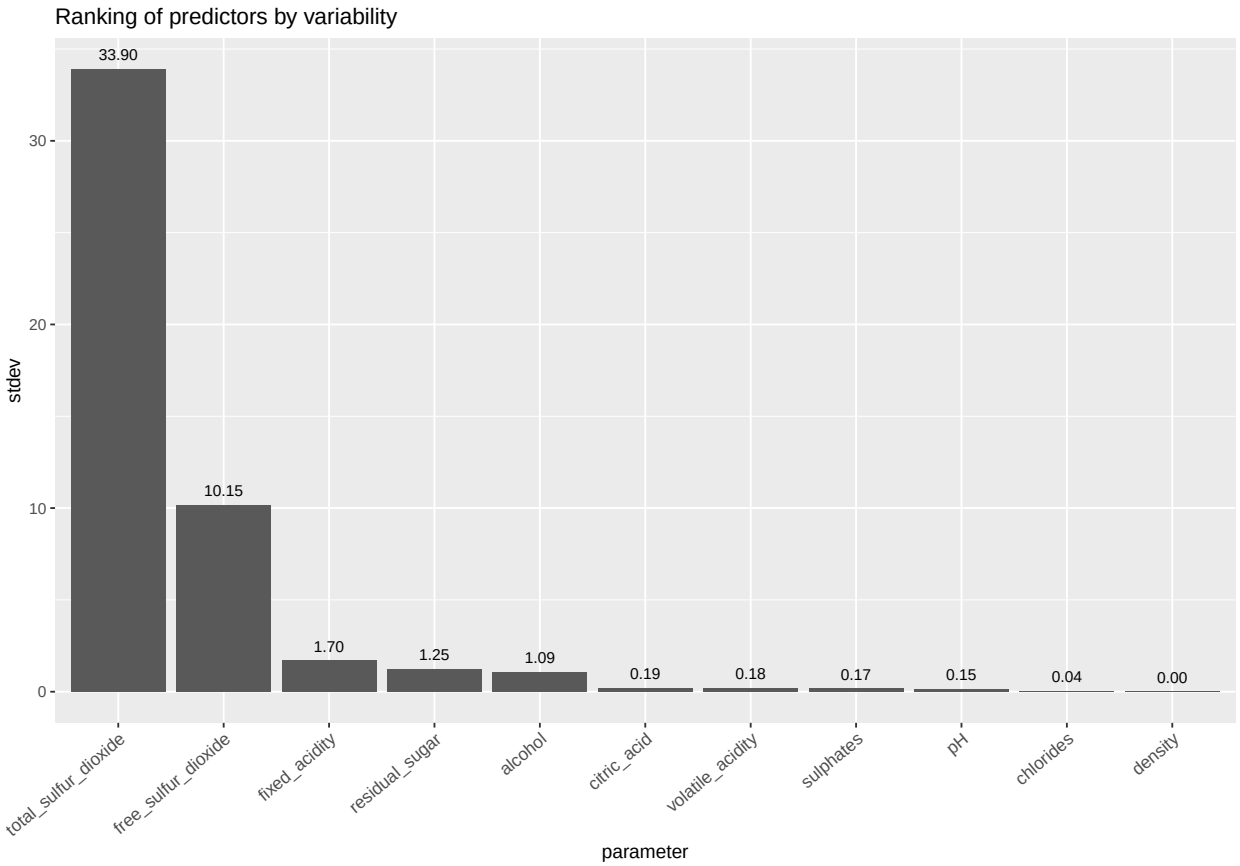
Next, we will take a look at the name of each chemical present at wine samples. We will also rank these chemicals for its variability within trainSet:

```
# variability analysis of predictors
X <- trainSet %>% select(-quality) %>% as.matrix()
sX <- colSds(X)
sX <- bind_cols(colnames(X), sX) %>% as_tibble()
```

```
## New names:
## * NA -> ...1
## * NA -> ...2
```

```
colnames(sX) <- c("parameter", "stdev")

sX %>%
  arrange(desc(stdev)) %>%    # First sort by val.
  mutate(parameter=factor(parameter, levels=parameter)) %>%   # This trick update the factor levels
  ggplot(aes(parameter, stdev)) +
  geom_bar(stat= "identity") +
  geom_text(aes(label=sprintf("%0.2f", stdev)), size=3, vjust=-0.8) +
  theme(axis.text.x = element_text(angle = 40, hjust = 1, size=10)) +
  ggtitle("Ranking of predictors by variability")
```



After checking the variability of all predictors, we will remove those that are considered as "low variability" by the "nearZeroVar" function.

It is worth to mention that, despite the fact the above histogram shows predictors with (visually) low variability, they are all kept on the database after executing "nearZeroVar". That means the function execution considered that those predictors do have enough variability to stay in.

```
# remove predictors with small variance
nzv <- trainSet %>%
  select(-quality) %>%
  nearZeroVar()
removedPredictors <- colnames(trainSet[,nzv])
removedPredictors
```

```
## character(0)
```

```
trainSet <- trainSet %>% select(-all_of(removedPredictors))
```

As a last step on the data preparation, memory is cleaned up and the outcome "quality" is changed to "factor", as all analysis to follow will be categorical (i.e. wine classification).

```
# clean for temporary data
rm(mainSet, testIndex, wineDF, X, p, p1, p2, sX)
rm(nzv, removedPredictors)

# change outcome to factor
trainSet$quality <- as_factor(trainSet$quality)
testSet$quality  <- as_factor(testSet$quality)
validationSet$quality <- as_factor(validationSet$quality)
```

Find the best model In this analysis, all but the first model (naive) will be checked and optimized against the trainSet, then tested using 10-fold cross validation against the testSet. Its correspondent Confusion Matrix and Accuracy, besides the key parameters for the best fit model, are displayed accordingly.

The naive average model This model simply takes the average rounded to its nearest integer of the "quality" vector within the trainSet as unique quality prediction for all wines. Due to this approach, one can anticipate its performance will be extremely poor. Nevertheless, the author decided to include it merely for didactic purposes.

```
# training
mu <- mean(as.numeric(trainSet$quality))
mu <- round(mu)

# predicting
N <- length(testSet$quality)
predicted <- replicate(N, mu) %>% factor(levels = qualityClasses)

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = testSet$quality)
cm$table
```

```
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  3  7 74 74 22  4
##          5  0  0  0  0  0  0
##          6  0  0  0  0  0  0
##          7  0  0  0  0  0  0
##          8  0  0  0  0  0  0
```

```
acc <- cm$overall["Accuracy"]

accuracyResults <- tibble(model    = "naive",
                          accuracy = acc)
accuracyResults
```

```
## # A tibble: 1 x 2
##   model accuracy
##   <chr>    <dbl>
## 1 naive   0.0380
```

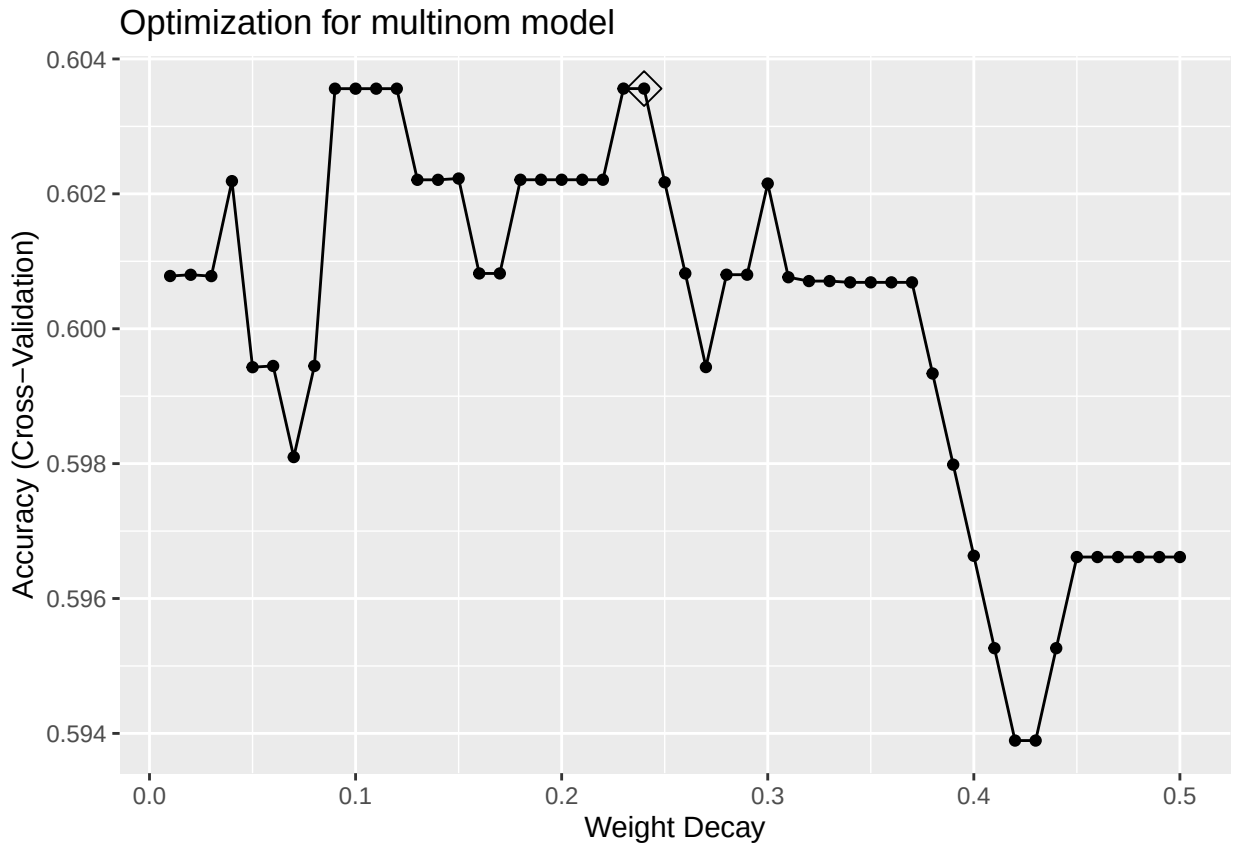
As anticipated, the prediction just assumes all wines are category 4, which leads to a poor Confusion Matrix, where its main diagonal is not dominated by high values as a good model would give. Its accuracy of only 3.8% also reflects that.

Penalized multinomial regression The model "multinom" has not been covered on course lectures. It is part of the "nnet" package and its documentation can be found at": https://cran.r-project.org/web/packages/nnet/nnet.pdf

Its tuneGrid has only one possible parameter (weight decay), and as such it is programmed to be scanned by the tuneGrid.

```
# training
set.seed(3, sample.kind="Rounding")
mNomFit <- trainSet %>% train(quality ~ .,
```

```
                              method = "multinom",
                              data = .,
                              trace = FALSE,    # suppress printing of iterations
                              tuneGrid = data.frame(decay = seq(0.01, 0.5, 0.01)),
                              trControl = control
)
plot_optimization_chart(mNomFit, "Optimization for multinom model")
```




```
## Call:
## nnet::multinom(formula = .outcome ~ ., data = dat, decay = param$decay,
##     trace = FALSE)
##
## Coefficients:
##   (Intercept) fixed_acidity volatile_acidity citric_acid residual_sugar
## 4    -0.25701      -0.30446          1.75241    -0.98685      0.1560550
## 5     1.52858      -0.17845          1.02142     0.63156      0.0065298
## 6     0.43662      -0.17767         -1.61394    -0.57791      0.0420458
## 7    -1.15409      -0.22777         -3.51690     0.48538      0.1286023
## 8    -0.79642      -0.73289         -0.15848     0.68902      0.3874453
##   chlorides free_sulfur_dioxide total_sulfur_dioxide  density       pH
## 4  -0.30368            0.043903              0.17550 -0.25505  1.05264
## 5   0.38874            0.036907              0.19160  1.53847  1.69989
## 6   0.48931            0.060400              0.17393  0.44576  0.46377
## 7  -0.77330            0.062280              0.17173 -1.17596 -1.51788
## 8  -0.15442           -0.046699              0.17419 -0.79554 -2.78797
##   sulphates  alcohol
```

```
## 4 -0.875551 -0.24561
## 5 -1.253149 -0.62628
## 6  0.046056  0.10540
## 7  1.731877  0.86117
## 8  0.678459  1.23335
##
## Residual Deviance: 1332.8
## AIC: 1452.8
```

```
# predicting
predicted <- testSet %>% predict(mNomFit, newdata = .)

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = testSet$quality)
cm$table
```

```
##            Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  3  4 58 18  1  0
##          6  0  2 14 50 12  2
##          7  0  1  2  6  9  2
##          8  0  0  0  0  0  0
```

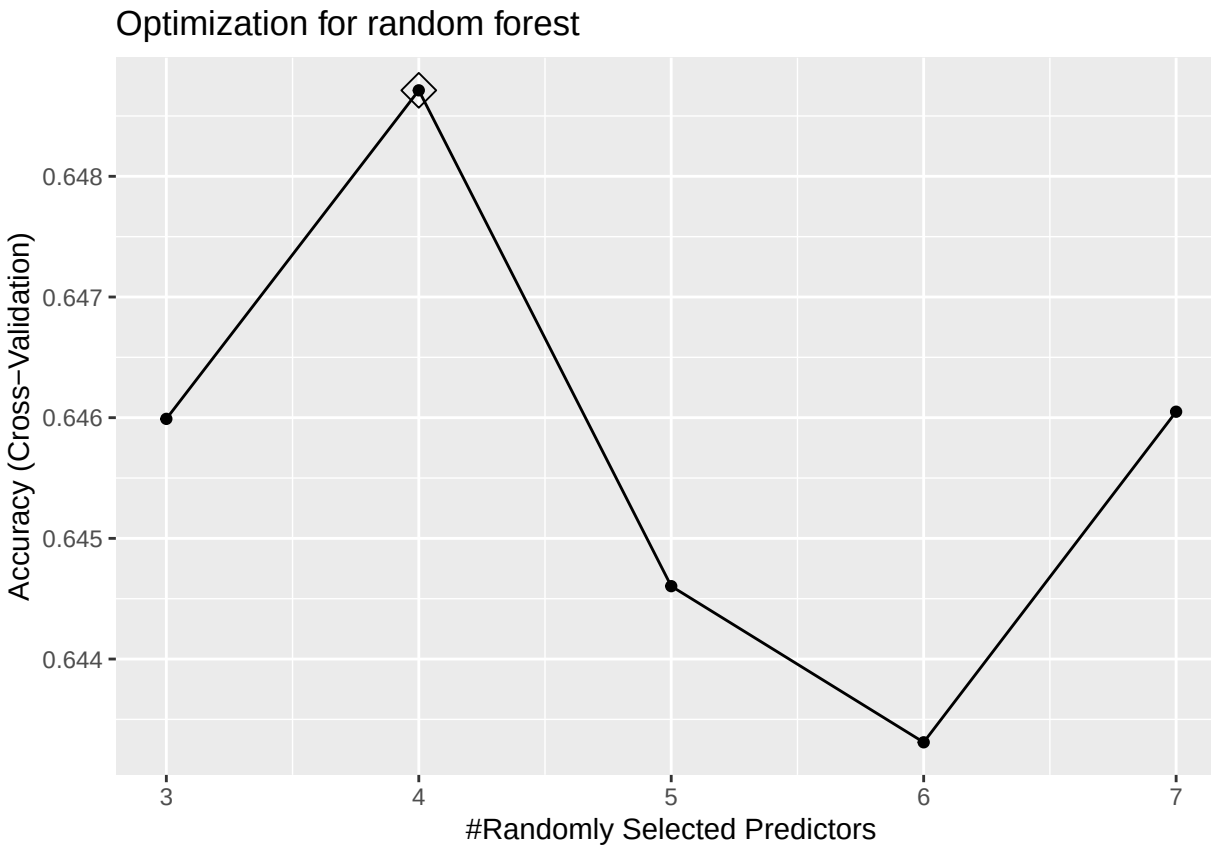
```
acc <- cm$overall["Accuracy"]

accuracyResults <- bind_rows(accuracyResults, tibble(model ="multinom",
                                                     accuracy = acc))
accuracyResults
```

```
## # A tibble: 2 x 2
##   model    accuracy
##   <chr>       <dbl>
## 1 naive      0.0380
## 2 multinom   0.636
```

Compared to the naive model, we have obtained a significant improvement on accuracy, above 60%. Its confusion matrix also present a much better prediction of categories 5-6, besides marginal prediction for category 7, and nonexistent prediction at categories 3, 4 and 8. The result is in line to the observed fact that there is a small amount of data for those nonperforming classes.

```
# training
set.seed(3, sample.kind="Rounding")
rfFit <- trainSet %>% train(quality ~ .,
                            method = "rf",
                            data = .,
                            tuneGrid = data.frame(mtry = seq(3, 7, 1)),
                            trControl = control)
plot_optimization_chart(rfFit, "Optimization for random forest")
```



Random forest

```
##
## Call:
##  randomForest(x = x, y = y, mtry = min(param$mtry, ncol(x)))
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 34.71%
## Confusion matrix:
##   3 4   5   6  7 8 class.error
## 3 0 0   1   1  0 0     1.00000
## 4 0 0  13   4  1 0     1.00000
## 5 0 0 235  76  2 0     0.24920
## 6 0 0  84 191 20 0     0.35254
## 7 0 0   5  39 50 1     0.47368
## 8 0 0   0   2  4 0     1.00000
```

```
# predicting
predicted <- testSet %>% predict(rfFit, newdata = ., type = "raw")

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = testSet$quality)
cm$table
```

```
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
```

```
##          4  0  0  0  0  0  0
##          5  3  5 57 17  1  0
##          6  0  2 15 52  8  2
##          7  0  0  2  5 12  2
##          8  0  0  0  0  1  0
```

```
acc <- cm$overall["Accuracy"]

accuracyResults <- bind_rows(accuracyResults, tibble(model ="rf",
                                                     accuracy = acc))
accuracyResults
```

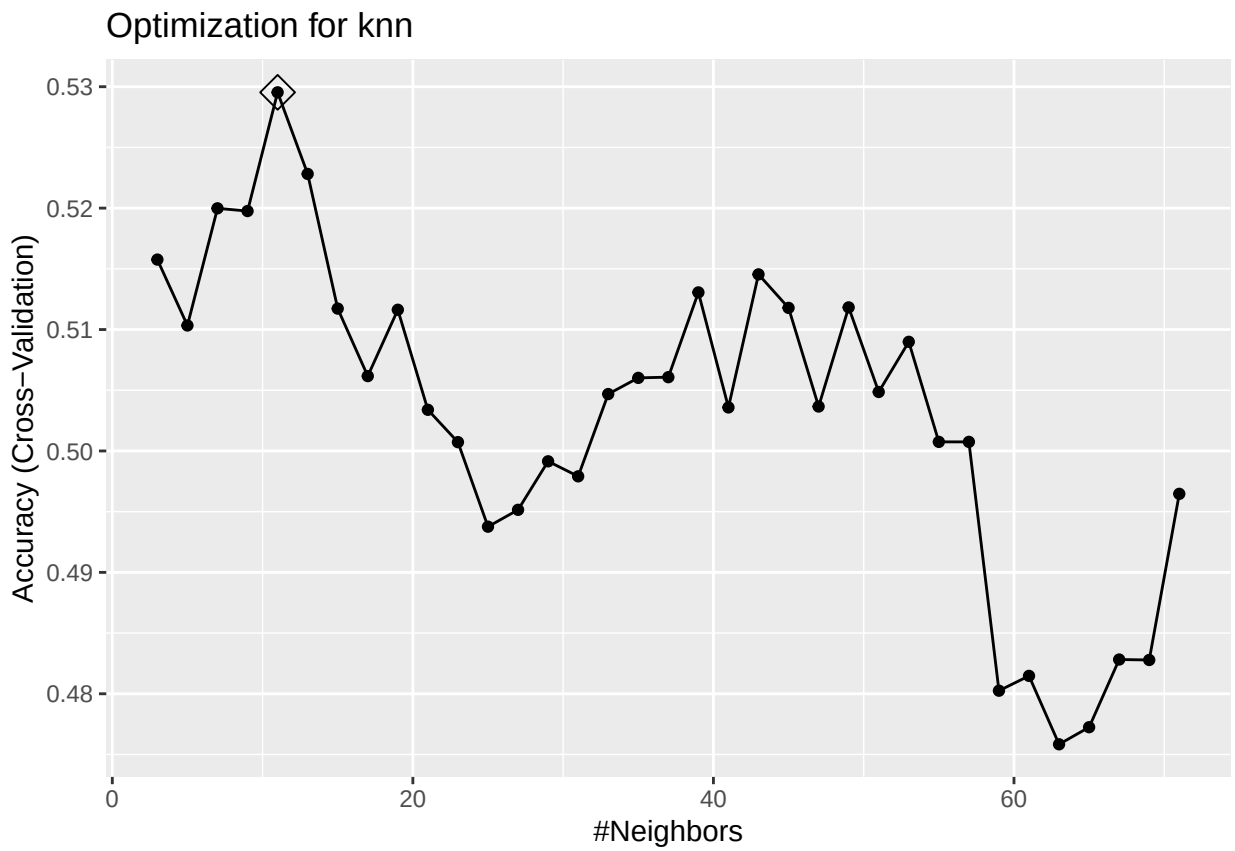
```
## # A tibble: 3 x 2
##   model    accuracy
##   <chr>       <dbl>
## 1 naive      0.0380
## 2 multinom   0.636
## 3 rf         0.658
```

The Random Forest model presented further improvement on overall accuracy than the previous Multinom model, apparently thanks to better prediction of category 7. However, prediction of categories 3, 4 and 8 remain unsatisfactory.

KNN This specific KNN (k-nearest-neighbors) model is part of the Caret package and it has been covered during course lectures. It has only one tuning parameter.

At a later step, we will also run the KKNN (weighted-KNN), which has more tuning parameters than this model.

```
# training
set.seed(3, sample.kind="Rounding")
knnFit <- trainSet %>% train(quality ~ .,
                             method = "knn",
                             data = .,
                             tuneGrid = data.frame(k = seq(3, 71, 2)),
                             trControl = control
)
plot_optimization_chart(knnFit, "Optimization for knn")
```




```
## 11-nearest neighbor model
## Training set outcome distribution:
##
##   3   4   5   6   7   8
##   2  18 313 295  95   6
```

```
# predicting
predicted <- testSet %>% predict(knnFit, newdata = ., type = "raw")

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = testSet$quality)
cm$table
```

```
##            Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  1  5 51 31  5  1
##          6  2  2 22 43 15  2
##          7  0  0  1  0  2  1
##          8  0  0  0  0  0  0
```

```
acc <- cm$overall["Accuracy"]

accuracyResults <- bind_rows(accuracyResults, tibble(model ="knn",
                                                     accuracy = acc))
accuracyResults
```

```
## # A tibble: 4 x 2
##   model    accuracy
##   <chr>       <dbl>
## 1 naive      0.0380
## 2 multinom   0.636
## 3 rf         0.658
## 4 knn        0.522
```

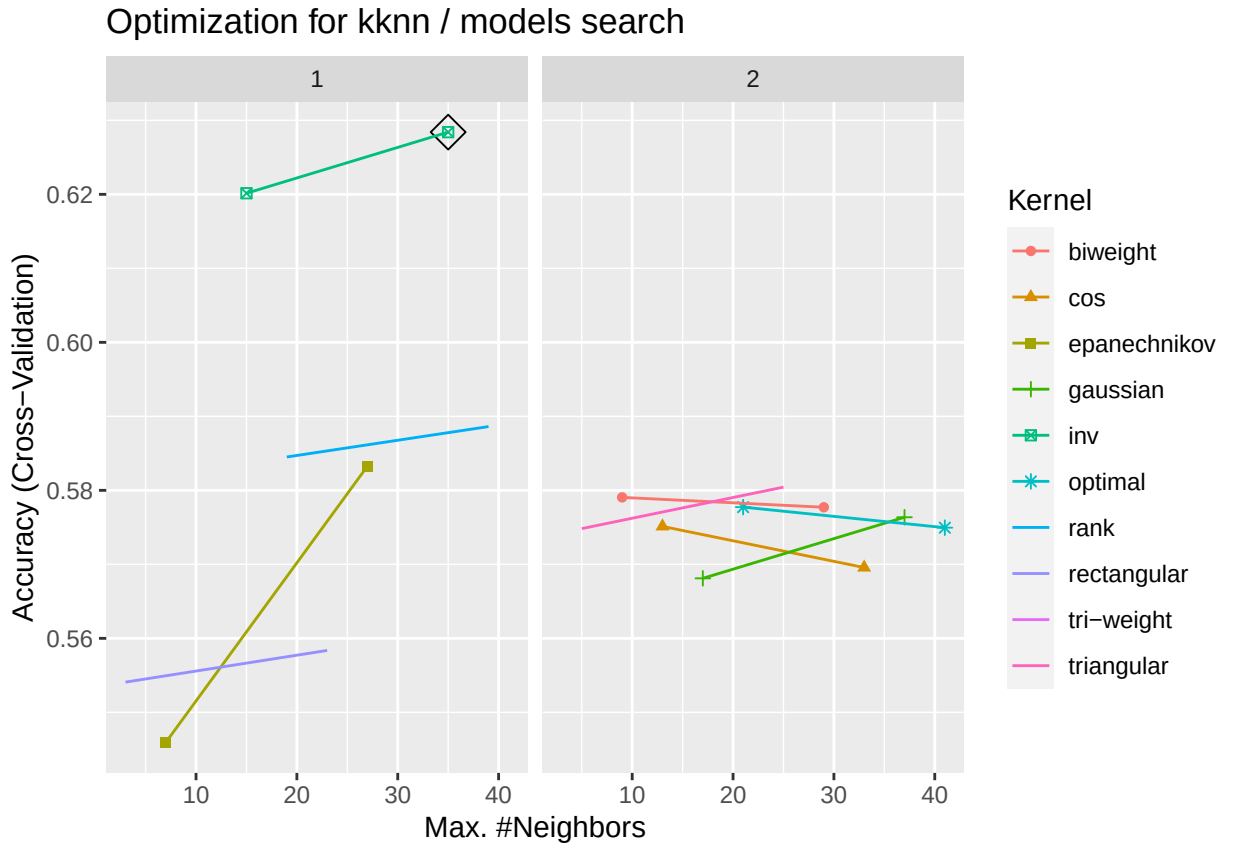
Accuracy for KNN is deteriorated, when compared to all past models but naive. This is reflected at the confusion matrix, which shows poor prediction at category 7.

KKNN / search among all models available The KKNN (weighted-KNN) model has three tuning parameters, one of them is the "kernel", which brings the possibility of analyzing a given data point taking into account its surrounding points as well.

For the tuning grid, the author decided to search the performance for all kernels. The best performing kernel will be taken for further tuning at the next step.

Model documentation can be found at: https://cran.r-project.org/web/packages/kknn/kknn.pdf

```
# training
set.seed(3, sample.kind="Rounding")
kknnFit <- trainSet %>% train(quality ~ .,
                              method = "kknn",
                              data = .,
                              tuneGrid = data.frame(kmax = seq(3, 41, 2),
                                                    distance = c(1, 2),
                                                    kernel = c("rectangular",
                                                               "triangular",
                                                               "epanechnikov",
                                                               "biweight",
                                                               "tri-weight",
                                                               "cos",
                                                               "inv",
                                                               "gaussian",
                                                               "rank",
                                                               "optimal"
                                                    )),
                              trControl = control)
plot_optimization_chart(kknnFit, "Optimization for kknn / models search")
```




```
##
## Call:
## kknn::train.kknn(formula = .outcome ~ ., data = dat, kmax = param$kmax,     distance = param$distanc
##
## Type of response variable: nominal
## Minimal misclassification: 0.36214
## Best kernel: inv
## Best k: 15
```

```
# predicting
predicted <- testSet %>% predict(kknnFit, newdata = ., type = "raw")

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = testSet$quality)
cm$table
```

```
##            Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  3  5 60 19  0  0
##          6  0  2 13 49  8  1
##          7  0  0  1  6 13  3
##          8  0  0  0  0  1  0
```

```
acc <- cm$overall["Accuracy"]
```

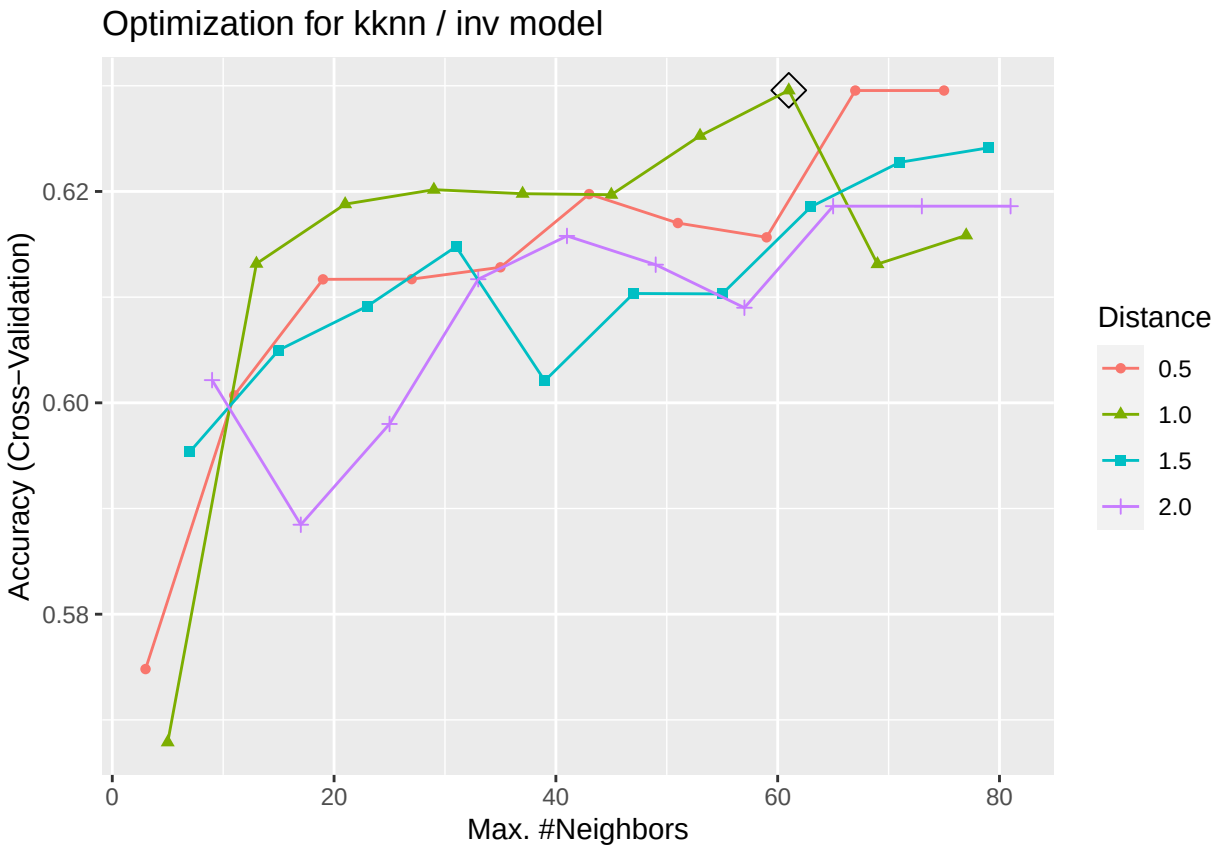
```
accuracyResults <- bind_rows(accuracyResults, tibble(model ="kknn/all models",
                                                     accuracy = acc))
accuracyResults
```

```
## # A tibble: 5 x 2
##   model           accuracy
##   <chr>              <dbl>
## 1 naive             0.0380
## 2 multinom          0.636
## 3 rf                0.658
## 4 knn               0.522
## 5 kknn/all models   0.663
```

So far, the KKNN is the best one analyzed. The optimization chart clearly shows the "inv" kernel brings the best performance compared to all others. For this reason, it will be further tuned on the next step.

KKNN / inv model The model "inv" is the winner from the search that was carried out above. As such, the tuning parameters for this run are expanded.

```
# training
set.seed(3, sample.kind="Rounding")
kknnInvFit <- trainSet %>% train(quality ~ .,
                                 method = "kknn",
                                 data = .,
                                 tuneGrid = data.frame(kmax = seq(3, 81, 2),
                                                       distance = seq(0.5, 2, 0.5),
                                                       kernel = "inv"),
                                 trControl = control)
plot_optimization_chart(kknnInvFit, "Optimization for kknn / inv model")
```




```
##
## Call:
## kknn::train.kknn(formula = .outcome ~ ., data = dat, kmax = param$kmax,     distance = param$distanc
##
## Type of response variable: nominal
## Minimal misclassification: 0.35802
## Best kernel: inv
## Best k: 61
```

```
# predicting
predicted <- testSet %>% predict(kknnInvFit, newdata = ., type = "raw")

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = testSet$quality)
cm$table
```

```
##            Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  3  6 62 16  1  0
##          6  0  1 12 58  9  2
##          7  0  0  0  0 12  2
##          8  0  0  0  0  0  0
```

```
acc <- cm$overall["Accuracy"]
```

```
accuracyResults <- bind_rows(accuracyResults, tibble(model ="kknn/inv",
                                                     accuracy = acc))
accuracyResults
```

```
## # A tibble: 6 x 2
##   model           accuracy
##   <chr>              <dbl>
## 1 naive             0.0380
## 2 multinom          0.636
## 3 rf                0.658
## 4 knn               0.522
## 5 kknn/all models   0.663
## 6 kknn/inv          0.717
```

We have obtained an accuracy in excess of 70% in the experiment. For this reason, the author chooses the kknn/inv model as the model for the final verification using the validationSet, at the next step.

```
# predicting
predicted <- validationSet %>% predict(kknnInvFit, newdata = ., type = "raw")

# accuracy calculation
cm <- confusionMatrix(data = predicted, reference = validationSet$quality)
cm$table
```

Best model validation

```
##            Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  0  0  0  0
##          5  1  4 68 23  0  0
##          6  0  4 28 64 18  5
##          7  0  0  0  6  8  1
##          8  0  0  0  0  0  0
```

```
acc <- cm$overall["Accuracy"]

accuracyResults <- bind_rows(accuracyResults, tibble(model ="kknn/inv/validation",
                                                     accuracy = acc))
accuracyResults
```

```
## # A tibble: 7 x 2
##   model                accuracy
##   <chr>                   <dbl>
## 1 naive                 0.0380
## 2 multinom              0.636
## 3 rf                    0.658
## 4 knn                   0.522
## 5 kknn/all models       0.663
## 6 kknn/inv              0.717
## 7 kknn/inv/validation   0.609
```

```
# reset warnings
options(warn = oldw)
```

The KKNN/Inv Kernel model is validated. The accuracy observed with the validation set has dropped to around 61%, from the previous 70+%. This result is expected for the fact that the validationSet was never seen by the model. As such, the performance of its prediction becomes lower than the one found when using the testSet (which has somehow been seen by the model).

Conclusion

The dataset WineQT was cleaned and analyzed. A number of models were tested, and the best performing one, the KKNN/Inv Kernel, showed and accuracy in excess of 70% on testing and above 60% on validation. A good generalization of the model has been achieved, thanks mainly to the 10-fold cross-validation executed on the learning/training phase.

On the other hand, lack of training data at categories 3, 4, 8 have led to a relatively poor performance on the accuracy of the model, despite the successful effort on improving the figures. The fact is confirmed by simple inspection of the confusion matrix.

Next steps / Future work

Two possible ways are foreseen by the author, in order to improve the model accuracy:

- To collect mode training data;
- To investigate other algorithms such as SVM and CNN.

The author is currently enrolled on a PhD program in Computer Science / AI / ML, with forecasted conclusion by 2024. He plans to research application of AI/ML on embedded microcontrollers, with and without GPU support. His plans include usage of the Julia language on his research, for the following reasons:

- It is a compiled language;
- Native parallel processing support;
- Growing support for ARM microcontrollers;
- Syntax similarity to Python.

References

- Text book: http://amlbook.com/
- NNET: https://cran.r-project.org/web/packages/nnet/nnet.pdf
- KKNN: https://cran.r-project.org/web/packages/kknn/kknn.pdf